# File System Management

*As used in PowerPC system version*

Thu, Mar 23, 2000

This note describes the usual practices of file system management in the PowerPC version of the system code.

***Uses of file system***

The earlier version of the system code that is based upon the 68K CPU family includes a home brew nonvolatile memory file system that serves as a repository mostly for local and page applications. When the system initializes, it automatically loads and starts execution of all resident enabled local applications. By this means, the enabled suite of local applications functions to extend the system code. Simple examples of local applications, which are actually structured as procedures called by the system code, include closed loops, Acnet task servers, and UDP port servers. The set of local applications need by each node can vary, although the system code is the same. Page applications are similar to local applications, but they have a simple page display user interface. Only one page application can be active at one time, but many local applications can be enabled and running at the same time. Although local applications do not support a user interface, one particular page application serves as a user interface for all local applications, in that it monitors the local application activity, supports configuration of each, and allows access to its parameters.

The Download page application supports a simple directory listing facility of the files in any node, as well as copying of files between nodes. This copy support preserves the version date of each program, so that it can serve to identify the actual version of a file that exists in any node. When a new version of a program is prepared on the development system, it is downloaded into one node via the TFTP protocol, perhaps for testing. This action establishes the version date of the file. As a matter of practice, if it is then desired to spread the new version to other nodes, the Download page is used to make the transfer, since using TFTP to target another node from the development system would establish a new version date for the same version of the file.

The contents of a program file in the 68K system is merely the raw position-independent code produced by the linker. Loading the code into memory for execution involves only a memory copy into allocated dynamic memory. Support for calls into system routines from local/page applications is supported via a TRAP facility. A short library stub executes a TRAP instruction with a routine number in a register; the address of that routine's entry point is found in a register upon return from the TRAP exception, whereupon a branch to subroutine is executed. Any arguments of the system routine were already placed on the stack by the calling program before the library stub was invoked.

***New system plan***

For the new PowerPC system, the process of invoking a routine in the system code is much more complicated. Under VxWorks, the program files are not really linked into executable code; the result of program preparation is only an object file that includes structures containing executable code, data, and external references. To load such a program file into memory for execution, the VxWorks "`ld`" command is used, which interprets the file format and places the code into one allocated area and the data into another area. Linkages to system routines cannot be dealt with in the same way as before. For this reason, the new system, rather than using a home brew file system, uses a RAM disk that is maintained under VxWorks as a DOS-format file system. In this way, the "`ld`" command can be used to bring a program into memory for execution. All the complexities of the PEF object file format will be interpreted by VxWorks. Linkages to system routines will be done automatically by the VxWorks dynamic loader. Each program file has external references to system routines only; none may have external references to other loaded files, as that would make downloading a new version of a single program problematic. (One would have to be sure to unload, and reload, all referenced file chains at the same time.)

The sequence of events in program preparation and installation for the new system is similar to that of the old, even if the program file format is different. The new feature that is added is to use the file version date that was established at the development system, so that multiple TFTP downloads of the same program file version will result in a consistent version date. This is accomplished by including the version date in the destination file name that is handed to the TFTP client. (This feature can also be added to the 68K system TFTP server support.)

*Internal file directory*

To assist the system code in managing the file system, the CODES system table provides a table directory facility. The format of a CODES table entry includes the following:

- 8-character file name
- file size
- nonvolatile memory address (68K only)
- new download flag
- execution address
- checksum (68K only)
- version date (calendar yr-mo-da-hr-mn-sc)
- usage counter (diagnostic)

File names are formed by concatenating a 4-character type name with a 4-character file name. The type name `LOOP` is used for local applications; `PAGE` is used for page applications; `DATA` is used for data files; and `HELP` is used for the text file that holds prompting information for local application parameters. Examples of 8-character file names are `LOOPGRAD`, which is a closed loop local application, and `PAGEMDMP`, which is the Memory Dump page application.

The "new download flag" is used to support the automatic switch to a new local application file version. If a new version of an enabled (executing) local application program is downloaded, the new download flag is set, and the system notices it to automatically invoke the previous version that is running (in dynamic memory) to terminate and release its resources, then perform the operation of loading the new version into memory and invoking it for initialization.

The execution address is used for invoking the program. For the PowerPC version, this will be the address of the transition vector which serves the same purpose of providing the means to invoke the program. It is important that this be efficient, since all enabled local application instances are invoked at 15Hz. (Instances refer to cases in which the same local application is invoked multiple times with different parameters and context.)

The version date identifies the version of the program that exists in the nonvolatile file system. Although it may exist as a part of the DOS-format file system in VxWorks nodes, it is provided here to be accessed via the usual Classic protocol methods.

***Version date***

The issue of maintaining a file version date is an important one. We need to keep the calendar date associated with each file as a documenting device so we can see what version is installed in any given node. The version date is in a 6-byte BCD format that specifies `yr`, `mo`, `da`, `hr`, `mn`, `sc` and is stored in a field of the associated CODES table entry with that file.

When a new file is transferred into a node using TFTP, the date associated with that file in the development system is obtained and formatted into the 6-byte version date form. An ascii representation of this date is appended to the file name to form the destination file name for use with the standard TFTP client-initiated transfer. The special TFTP server on the target node notices the 8-character file name followed by an ascii version date. It strips off the version date from the name and opens a RAM disk file for writing the data that is passed from the client. The version date is then recorded in the CODES table entry at the conclusion of the transfer. The RAM disk file system also carries a date, and this can be made the same, even though we may not be able to simply access it via commands at a VxWorks prompt.

So, a special TFTP server is required that interprets file names with optional version dates appended. (Without an appended version date, it will simply use the current time-of-day as a version date.) Special file name formats allow the server to recognize and permit access to various system tables and even to general system memory.

As to what form the appended version date might take, suppose we have an example of a page application for the memory dump display, which has the 8-character name `PAGEMDMP`. Its PEF file name on the development system is `PAGEMDMP.o`. The variation of this name sent to a target node via TFTP protocol may be `PAGEMDMP.o.000322144906`. The appended version date would consist of a dot and 12 decimal characters that signify the version date in the form `YrMoDaHrMnSc`. The total file name length would be 8+3+12, or 23 characters.

Note that the development system file name can be `PAGEMDMP.o`. For the TFTP transfer, this is used as the source file name; this same name appended with the ascii form of the version date is used as the destination file name.

***Real time concern***

It is currently possible to download a new version of a program file without impacting the real time (15Hz) performance of the front end. The VxWorks version of the system will necessarily use RAM disk software that will presumably operate more slowly. We will have to see what effect downloading has on an operating VxWorks system. There is no backup plan. If it should cause problems for which we cannot devise a work-around, we shall have to refrain from updating such files during accelerator operations, just as we refrain from updating system code today, which necessitates a reboot that is usually unpopular with operations personnel.